

CONJUGATE GRADIENT WITH SUBSPACE OPTIMIZATION

SAHAR KARIMI AND STEPHEN VAVASIS

Abstract. In this paper we present a variant of the conjugate gradient (CG) algorithm in which we invoke a subspace minimization subproblem on each iteration. We call this algorithm CGSO for “conjugate gradient with subspace optimization”. It is related to earlier work by Nemirovsky and Yudin. We apply the algorithm to solve unconstrained strictly convex problems. As with other CG algorithms, the update step on each iteration is a linear combination of the last gradient and last update. Unlike some other conjugate gradient methods, our algorithm attains a theoretical complexity bound of $O(\sqrt{L/l} \log(1/\epsilon))$, where the ratio L/l characterizes the strong convexity of the objective function. In practice, CGSO competes with other CG-type algorithms by incorporating some second order information in each iteration.

Key words. Conjugate Gradient, First Order Algorithms, Convex Optimization

AMS subject classifications.

1. Introduction. The method of conjugate gradients (CG) was introduced by Hestenes and Stiefel in 1952 [8] to minimize quadratic objective functions of the form $f(\mathbf{x}) = \mathbf{x}^t A \mathbf{x} / 2 - \mathbf{b}^t \mathbf{x}$ (or equivalently, to solve $A \mathbf{x} = \mathbf{b}$), where A is a symmetric positive definite matrix. We refer to this algorithm as “linear conjugate gradient.” Later, the algorithm was generalized to “nonlinear conjugate gradient” which addresses unconstrained minimization of arbitrary differentiable objective functions, by Fletcher and Reeves [3] and Polak and Ribière [13], and others. Nonlinear CG algorithms all have the property that when applied to a quadratic objective function and coupled with an exact line search, they reduce to linear CG. None of these algorithms, however, has a known complexity bound when applied to nonquadratic functions. Indeed, Nemirovsky and Yudin [9] argue that their worst-case complexity for strictly convex functions is quite poor.

Nemirovsky and Yudin, on the other hand, found a different generalization of CG. Actually, their algorithm, which we refer to as NYCG, is not a generalization of the CG algorithm itself but rather of its derivation from some equations and inequalities that underlie it. Their algorithm achieves a worst-case complexity bound of $O(\ln(1/\epsilon) \sqrt{L/l})$. Here ϵ is the desired relative accuracy, that is, $\epsilon = (f(\mathbf{x}^n) - f(\mathbf{x}^*)) / (f(\mathbf{x}^0) - f(\mathbf{x}^*))$, where \mathbf{x}^0 is the starting point, \mathbf{x}^* is the optimizer, and \mathbf{x}^n is the final iterate, and L and l characterize the convexity of f . In particular, we assume that for all \mathbf{x}, \mathbf{y} lying in the level set of \mathbf{x}^0 ,

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L \|\mathbf{x} - \mathbf{y}\|, \quad (1.1)$$

$$f(\mathbf{y}) - f(\mathbf{x}) \geq \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{l}{2} \|\mathbf{y} - \mathbf{x}\|^2. \quad (1.2)$$

For example, in the case of a convex quadratic function $f(\mathbf{x}) = \mathbf{x}^t A \mathbf{x} / 2 - \mathbf{b}^t \mathbf{x}$, L/l is the condition number of A . From inequality (1.1), it follows that

$$f(\mathbf{y}) - f(\mathbf{x}) \leq \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{L}{2} \|\mathbf{y} - \mathbf{x}\|^2, \quad (1.3)$$

which will be useful in our analysis.

One drawback of the NYCG algorithm is that in the case of quadratic objective functions, it does not reduce to linear CG (and in fact, is much slower in practice). A second drawback is that it requires prior knowledge of the ratio L/l . This is because

the algorithm needs to be restarted every $O(\sqrt{L/l})$ iterations in order to achieve the above-mentioned complexity bound.

A further disadvantage of NYCG is that it has a relatively expensive computation on every iteration, namely, one must solve a two-dimensional convex optimization problem using the ellipsoid method. This drawback was later remedied by a different algorithm due to Nesterov [10]. Nesterov’s algorithm generates two sequences of iterates; the first sequence of iterates is generated such that a sufficient reduction in objective function is achieved, and the new iterate in the second sequence is an affine combination of the last two iterates of the first sequence. Nesterov’s algorithm, however, still has the two disadvantages mentioned in the previous paragraph.

The purpose of this paper is to develop an algorithm that we call conjugate gradient with subspace optimization (CGSO) that mainly avoids the disadvantages mentioned above. CGSO is mostly closely related to NYCG. In particular, we seek a conjugate-gradient-like algorithm with the following properties.

1. The algorithm should reduce to linear CG when the objective function is quadratic.
2. The algorithm should have the complexity bound of $O(\ln(1/\epsilon)\sqrt{L/l})$ iterations for convex functions satisfying (1.1) and (1.2).
3. The algorithm should not require prior knowledge of any parameters describing f .
4. The cost per iteration should not be excessive.

The algorithm we propose achieves goals 1–3, and mostly achieves goal 4. Like NYCG, our algorithm must solve a convex optimization subproblem on every iteration. The dimension of the subproblem is always at least 2 and is determined adaptively by the algorithm. The worst-case upper bound we are able to prove for the dimension of this subproblem is $O(\log j)$, where j is the number of iterations so far. However, in our testing, the dimension of the subproblem was 2 in almost every case; in one test case it reached the value 3 for a few iterations, but it never exceeded 3. Furthermore, we find that solving the subproblem is usually fairly efficient because we usually apply Newton’s method, using automatic differentiation to obtain the necessary first and second derivatives.

Goal 3, the condition of no prior knowledge of parameters, has the obvious benefit of making the algorithm easier to apply in practice. It also has a second subtler benefit. For some convex problems, e.g., minimization of a log-barrier function, there is no prior bound on L/l over the domain of the function since the derivatives tend to infinity at the boundaries. However, as the minimizer is approached, the bad behavior at the boundaries becomes irrelevant, and there is a new smaller ratio L/l relevant for level sets in the neighborhood of the minimizer. In this case, CGSO automatically adapts to the improved value of L/l . Such adaptation is possible also with NYCG, and Nesterov’s algorithm too, but, as far as we know, the adaptation must be done by the user and cannot be easily automated.

Methods reviewed in this paper are among techniques that are generally referred to as “first-order algorithms”, because they use only the first derivative information of the function in each iteration. Due to the successful theory of NYCG and Nesterov techniques, first-order algorithms have attracted many researchers during the last decade and have been extended to solving different classes of problems. Nesterov in [11] proposed a variation of his earlier algorithms for minimizing a nonsmooth function. In addition to nonsmooth optimization, Nesterov algorithm has been adapted for constrained problems with simple enough feasible regions so that a projection on these

sets can be easily computed. One may refer to [15] and references therein for a more elaborated discussion on different adaptations of Nesterov's algorithm. The focus of this paper, however, is more on CG algorithm and not on first-order techniques in general.

We now provide further background on conjugate gradient. The original linear CG has the following form:

$$\mathbf{x}^{j+1} = \mathbf{x}^j - \frac{(\mathbf{r}^j)^t \mathbf{d}^j}{(\mathbf{d}^j)^t \mathbf{A} \mathbf{d}^j}, \quad (1.4a)$$

$$\mathbf{d}^{j+1} = -\mathbf{r}^{j+1} + \frac{(\mathbf{r}_{j+1})^t \mathbf{A} \mathbf{d}^j}{(\mathbf{d}^j)^t \mathbf{A} \mathbf{d}^j} \mathbf{d}^j. \quad (1.4b)$$

In the above equations \mathbf{r}^j is $\nabla f(\mathbf{x}) = \mathbf{A} \mathbf{x}^j - \mathbf{b}$ and $\mathbf{d}^0 = -\mathbf{r}^0$. It is possible to show that the number of iterations in linear CG is bounded by the dimension of the problem, n . For more details on linear CG, one may refer to [5] or [12].

Nonlinear CG was proposed by Fletcher and Reeves [3] as an adaptation of the above algorithm for minimizing a general nonlinear function. The general form of this algorithm is as follows:

$$\mathbf{x}^{j+1} = \mathbf{x}^j + \alpha^j \mathbf{d}^j, \quad (1.5a)$$

$$\mathbf{d}^{j+1} = -\mathbf{g}^{j+1} + \beta^j \mathbf{d}^j. \quad (1.5b)$$

Here, \mathbf{d}^j is the *search direction* at each iteration, \mathbf{g}^{j+1} is the gradient of the function at $(j+1)$ th iterate, i.e., $\nabla f(\mathbf{x}^{j+1})$; and α^j is the step size, usually determined by a line search. Different updating rules for β^j give us different variants of nonlinear CG. The most common formulas for computing β^j are:

$$\begin{aligned} \text{Fletcher-Reeves (1964): } \beta_{FR} &= \frac{\|\mathbf{g}^{j+1}\|}{\|\mathbf{g}^j\|}, \\ \text{Polak-Ribière (1969): } \beta_{PR} &= \frac{(\mathbf{g}^{j+1})^t (\mathbf{g}^{j+1} - \mathbf{g}^j)}{\|\mathbf{g}^j\|^2}. \end{aligned}$$

Hager and Zhang [6] present a complete list of all updating rules in their survey on nonlinear CG. The convergence of nonlinear CG is highly dependent on the line search; for some the exact line search is crucial. There are numerous papers devoted to the study of global convergence of nonlinear CG algorithms, most of which discuss variants of nonlinear CG that do not rely on exact line search to be globally convergent. Al-Baali [1] discusses the convergence of Fletcher-Reeves algorithm with exact line search. Gilbert and Nocedal [4] study the convergence of nonlinear CG algorithms with no restart and no exact line search. Dai and Yuan [2] present a nonlinear CG for which the standard Wolfe condition suffices. A recent variant of CG has been proposed by Hager and Zhang [7] that relies on a line search satisfying the Wolfe Conditions. Furthermore this algorithm has the advantage that every search direction is a descent direction, which is not necessarily the case in nonlinear CG.

From Yuan and Stoer's perspective [16], CG is a technique in which the search direction \mathbf{d}^{j+1} lies in the subspace spanned by $\text{Sp}\{\mathbf{g}^j, \mathbf{d}^j\}$. In the algorithm they propose they compute the new search direction by minimizing a quadratic approximation of the objective function over the mentioned subspace. A more generalized

form of CG called Heavy Ball Method, was introduced by Polyak [14], in which \mathbf{x}^{j+1} is $\mathbf{x}^j + \alpha(-\mathbf{g}^j) + \beta(\mathbf{x}^j - \mathbf{x}^{j-1})$. He proved a geometric progression rate for this algorithm when α and β belong to a specific range. In the same work, Polyak reviews the CG method; our simplest form of CGSO, which is given in section 2.1, coincides with his presentation of CG.

The remainder of this paper is devoted to CGSO and its properties. In section 2 we describe the algorithm. Our main result on the convergence of the algorithm is presented in section 2.3. The implementation of the algorithm is discussed in section 3; furthermore the results and comparison of CGSO with CG are presented in this section.

2. CGSO. In this section we present CGSO for solving the problem

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}), \quad (2.1)$$

where $f(\mathbf{x})$ is a strictly convex function characterized by parameters L and l . We follow the standard notation throughout this paper. $\langle \cdot, \cdot \rangle$ represents the inner product of two vectors in proper dimension, and $\|\cdot\|$ stands for the 2-norm of a vector unless otherwise is stated. Bold lower case characters and upper case characters are used for vectors and matrices respectively; and their superscript states the iteration count.

2.1. The Algorithm. Let $g(\mathbf{x})$ denote the derivative of $f(\mathbf{x})$; CGSO, in its preliminary form, is as follows:

- $\mathbf{x}^0 = \text{arbitrary}$;
- for $j = 1, 2, \dots$
 - $\mathbf{x}^{j+1} = \mathbf{x}^j + \alpha^j g(\mathbf{x}^j) + \beta^j \mathbf{d}^j$ where
 - $\mathbf{d}^j = \mathbf{x}^j - \mathbf{x}^{j-1}$ and
 - $\alpha^j, \beta^j = \text{argmin}_{\alpha, \beta} f(\mathbf{x}^j + \alpha g(\mathbf{x}^j) + \beta \mathbf{d}^j)$

As we shall see, some modifications are necessary to achieve the desired complexity bound. The above algorithm is certainly a generalization of nonlinear CG, since each search direction is a combination of previous gradients. Notice that the above algorithm is a descent method (i.e. $f(\mathbf{x}^{j+1}) \leq f(\mathbf{x}^j)$). Furthermore, the above algorithm performs at least as well as steepest descent in each iteration; hence it converges to a stationary point, which, for the class of convex problems, coincides with the minimizer of the function.

For convenience, let us represent $g(\mathbf{x}^j)$ by \mathbf{g}^j , and let $v_f(\mathbf{x})$ denote the residual of the function, i.e. $f(\mathbf{x}) - f(\mathbf{x}^*)$. By strong convexity of the function, we get the following properties for the above algorithm:

- (a) $f(\mathbf{x}^{j+1}) \leq f(\mathbf{x}^j) - \frac{1}{2L} \|\mathbf{g}^j\|^2$
- (b) $\langle \mathbf{g}^j, \mathbf{x}^* - \mathbf{x}^j \rangle \leq f(\mathbf{x}^*) - f(\mathbf{x}^j)$
- (c) $v_f(\mathbf{x}^0) = f(\mathbf{x}^0) - f^* \geq \frac{l}{2} (\mathbf{x}^* - \mathbf{x}^0)^2$

Property (a) follows from (1.3) and the fact that $f(\mathbf{x}^{j+1}) \leq f(\mathbf{x}^j - \frac{1}{L} \mathbf{g}^j)$. Property (b) is true by convexity of the function, and property (c) is a direct derivation from inequality (1.2). We are now ready to present the main lemma on the complexity bound of the algorithm.

LEMMA 2.1. Suppose $m \geq \left\lceil 8\rho\sqrt{\frac{L}{l}} \right\rceil$ and

$$\frac{(f(\mathbf{x}^{m-1}) - f(\mathbf{x}^0))}{4} \left(\sum_{j=0}^{m-1} \lambda^j \right) + \sum_{j=0}^{m-1} \lambda^j \langle \mathbf{g}^j, \mathbf{x}^j - \mathbf{x}^0 \rangle < 0, \quad (2.2)$$

and

$$\left\| \sum_{j=0}^{m-1} \lambda^j \mathbf{g}^j \right\| \leq \rho \sqrt{\sum_{j=0}^{m-1} (\lambda^j)^2 \|\mathbf{g}^j\|^2}, \quad (2.3)$$

are satisfied, where ρ is a constant ≥ 1 , and

$$\lambda^j = \sqrt{\frac{f(\mathbf{x}^j) - f(\mathbf{x}^{j+1})}{\|\mathbf{g}^j\|^2}}$$

. Then the residual of the function is divided in half after m iterations; i.e. $v_f(\mathbf{x}^m) \leq \frac{1}{2}v_f(\mathbf{x}^0)$.

Proof. Our proof is an extension of the proof in section 7.3 in [9]. Suppose by contradiction that $m \geq \left\lceil 8\rho\sqrt{\frac{L}{t}} \right\rceil$, (2.2) and (2.3) are satisfied; but $v_f(\mathbf{x}^m) > \frac{v_f(\mathbf{x}^0)}{2}$.

By definition of λ^j ,

$$f(\mathbf{x}^{j+1}) = f(\mathbf{x}^j) - (\lambda^j)^2 \|\mathbf{g}^j\|^2,$$

hence

$$v_f(\mathbf{x}^{j+1}) = v_f(\mathbf{x}^j) - (\lambda^j)^2 \|\mathbf{g}^j\|^2.$$

Summing these inequalities over $j = 0, \dots, m-1$, we get:

$$0 \leq v_f(\mathbf{x}^m) = v_f(\mathbf{x}^0) - \sum_{j=0}^{m-1} (\lambda^j)^2 \|\mathbf{g}^j\|^2,$$

or equivalently,

$$\sum_{j=0}^{m-1} (\lambda^j)^2 \|\mathbf{g}^j\|^2 \leq v_f(\mathbf{x}^0). \quad (2.4)$$

By convexity of the function we have,

$$\langle \mathbf{g}^j, \mathbf{x}^* - \mathbf{x}^j \rangle \leq f(\mathbf{x}^*) - f(\mathbf{x}^j) = -v_f(\mathbf{x}^j),$$

and so

$$\langle \mathbf{g}^j, \mathbf{x}^* - \mathbf{x}^0 \rangle - \langle \mathbf{g}^j, \mathbf{x}^j - \mathbf{x}^0 \rangle \leq -v_f(\mathbf{x}^j) \leq -v_f(\mathbf{x}^m) < \frac{-v_f(\mathbf{x}^0)}{2}.$$

Let's consider the weighted sum of all the above inequalities for $j = 0, \dots, m-1$ with weights λ^j 's to get:

$$\left\langle \sum_{j=0}^{m-1} \lambda^j \mathbf{g}^j, \mathbf{x}^* - \mathbf{x}^0 \right\rangle - \sum_{j=0}^{m-1} \lambda^j \langle \mathbf{g}^j, \mathbf{x}^j - \mathbf{x}^0 \rangle < \frac{-v_f(\mathbf{x}^0)}{2} \left(\sum_{j=0}^{m-1} \lambda^j \right),$$

which can be rearranged to the following form,

$$\left\langle \sum_{j=0}^{m-1} \lambda^j \mathbf{g}^j, \mathbf{x}^* - \mathbf{x}^0 \right\rangle < -\frac{v_f(\mathbf{x}^0)}{2} \left(\sum_{j=0}^{m-1} \lambda^j \right) + \sum_{j=0}^{m-1} \lambda^j \langle \mathbf{g}^j, \mathbf{x}^j - \mathbf{x}^0 \rangle.$$

Equivalently we can rewrite the above inequality as:

$$\begin{aligned} \left\langle \sum_{j=0}^{m-1} \lambda^j \mathbf{g}^j, \mathbf{x}^* - \mathbf{x}^0 \right\rangle &< -\frac{v_f(\mathbf{x}^0)}{4} \left(\sum_{j=0}^{m-1} \lambda^j \right) \\ &+ \left(\frac{f(\mathbf{x}^*) - f(\mathbf{x}^0)}{4} \left(\sum_{j=0}^{m-1} \lambda^j \right) + \sum_{j=0}^{m-1} \lambda^j \langle \mathbf{g}^j, \mathbf{x}^j - \mathbf{x}^0 \rangle \right). \end{aligned}$$

Using inequality (2.2) along with the facts that $f(\mathbf{x}^*) \leq f(\mathbf{x}^j)$ and $\lambda^j \geq 0$ for all j , we get:

$$\left\langle \sum_{j=0}^{m-1} \lambda^j \mathbf{g}^j, \mathbf{x}^* - \mathbf{x}^0 \right\rangle < -\frac{v_f(\mathbf{x}^0)}{4} \left(\sum_{j=0}^{m-1} \lambda^j \right). \quad (2.5)$$

By the Cauchy-Schwarz inequality we have

$$-\left\| \sum_{j=0}^{m-1} \lambda^j \mathbf{g}^j \right\| \|\mathbf{x}^* - \mathbf{x}^0\| \leq \left\langle \sum_{j=0}^{m-1} \lambda^j \mathbf{g}^j, \mathbf{x}^* - \mathbf{x}^0 \right\rangle < -\frac{v_f(\mathbf{x}^0)}{4} \left(\sum_{j=0}^{m-1} \lambda^j \right),$$

hence

$$\left\| \sum_{j=0}^{m-1} \lambda^j \mathbf{g}^j \right\| \|\mathbf{x}^* - \mathbf{x}^0\| > \frac{v_f(\mathbf{x}^0)}{4} \left(\sum_{j=0}^{m-1} \lambda^j \right). \quad (2.6)$$

By property (c) we have

$$\|\mathbf{x}^* - \mathbf{x}^0\| \leq \sqrt{\frac{2v_f(\mathbf{x}^0)}{l}}. \quad (2.7)$$

Furthermore, by inequalities (2.3) and (2.4) we get:

$$\left\| \sum_{j=0}^{m-1} \lambda^j \mathbf{g}^j \right\| \leq \rho \sqrt{\sum_{j=0}^{m-1} (\lambda^j)^2 \|\mathbf{g}^j\|^2} \leq \rho \sqrt{v_f(\mathbf{x}^0)}. \quad (2.8)$$

Replacing inequalities (2.7) and (2.8) in inequality (2.6), we get

$$\rho \sqrt{v_f(\mathbf{x}^0)} \sqrt{\frac{2v_f(\mathbf{x}^0)}{l}} > \frac{v_f(\mathbf{x}^0)}{4} \left(\sum_{j=0}^{m-1} \lambda^j \right). \quad (2.9)$$

Notice that by definition of λ and property (a), $\lambda^j \geq \sqrt{\frac{1}{2L}}$ for all j , so

$$\sum_{j=0}^{m-1} \lambda^j \geq \sqrt{\frac{1}{2L}} m.$$

Using this fact in inequality (2.9), we get

$$\rho \sqrt{v_f(\mathbf{x}^0)} \sqrt{\frac{2v_f(\mathbf{x}^0)}{l}} > \frac{v_f(\mathbf{x}^0)}{4} \left(\sqrt{\frac{1}{2L}} m \right),$$

therefore

$$m < 8\rho\sqrt{\frac{L}{l}}, \quad (2.10)$$

which contradicts our assumption on the value of m . \square

Lemma 2.1 shows that under conditions (2.2) and (2.3), the residual of the function is divided in half every $m = O(\sqrt{\frac{L}{l}})$ iterations. For the next sequence of m iterations, a further reduction of $\frac{1}{2}$ is achieved provided (2.2) and (2.3) hold, with \mathbf{x}^m substituted in place of \mathbf{x}^0 . Hence by letting \mathbf{x}^m be the new \mathbf{x}^0 and repeating the same algorithm, we can find the ϵ -optimal solution in $\lceil \log_2 \frac{1}{\epsilon} \rceil \lceil 8\rho\sqrt{\frac{L}{l}} \rceil$ iterations. Of course, m is not known to our algorithm, a point to which we return in the next section.

2.2. Restarting. As mentioned before, the algorithm should use first 0, then m , then $2m$ and so on when checking (2.2) and (2.3) in order to achieve the desired complexity bound. We use the term “restarting” to refer to the process of replacing \mathbf{x}^0 in inequalities (2.2) and (2.3) with \mathbf{x}^m for some $m > 0$. Notice that this process does not change any of the iterates, and it only changes the interval of indices in which we check inequalities (2.2) and (2.3).

If we know parameters of the function, i.e. L and l , we can compute m directly; hence we would be able to determine exactly when we need to restart the algorithm. In many cases, however, finding the parameters of the function is a nontrivial problem itself. In order to have an algorithm which does not rely on any prior knowledge about the function, we propose the following technique.

Since $2^p \leq m \leq 2^{p+1}$ for some p , by restarting the algorithm every 2^{p+1} iterations, we are guaranteed that the residual of the function is divided in half between every two consecutive restarts which contains at most $2m = \lceil 16\rho\sqrt{\frac{L}{l}} \rceil$ iterations, where the last statement follows from the fact that $2^{p+1} \leq 2m$. Since the exact value of p is usually unknown, we repeat the above procedure for every value of $p \in \{P_l, \dots, P_u\}$. In the section on implementation of the algorithm we will discuss the range of p we used, however note that P_u does not need to exceed $\lceil \log_2 j \rceil$, where j is the current iteration count. This is because inequalities (2.2) and (2.3) are the same for all $p \geq \lceil \log_2 j \rceil$. We can now state the CGSO algorithm in a more complete form:

\mathbf{x}^0 = arbitrary; ρ : given;

For $j = 1, 2, \dots$

$\mathbf{x}^{j+1} = \operatorname{argmin}_{\mathbf{x} \in E^j} f(\mathbf{x})$ where $E^j = \mathbf{x}^j + \operatorname{Sp}\{\mathbf{g}^j, \mathbf{d}^j\}$

Let $\lambda^j = \sqrt{\frac{f(\mathbf{x}^j) - f(\mathbf{x}^{j+1})}{\|\mathbf{g}^j\|^2}}$

For $p = P_l, \dots, \lceil \log_2 j \rceil$

If $j+1 = k_p 2^p$ for some integer k_p

Let $r_p = (k_p - 1) 2^p$; check

$$\frac{(f(\mathbf{x}^{j+1}) - f(\mathbf{x}^{r_p}))}{4} \left(\sum_{i=r_p}^j \lambda^i \right) + \sum_{i=r_p}^j \lambda^i \langle \mathbf{g}^i, \mathbf{x}^i - \mathbf{x}^{r_p} \rangle < 0$$

and

$$\left\| \sum_{i=r_p}^j \lambda^i \mathbf{g}^i \right\| \leq \rho \sqrt{\sum_{i=r_p}^j (\lambda^i)^2} \|\mathbf{g}^i\|^2$$

If any of the above inequalities fails, take the “correction step”.

(which will be defined in the subsequent section)

2.3. Correction Step. Let us refer to the set of iterates between two consecutive restart as a “block” of iterates; in other words for any p , the set of iterates $\mathbf{x}^0, \mathbf{x}^1, \dots, \mathbf{x}^{2^p-1}$ is the first block of size 2^p , $\mathbf{x}^{2^p}, \mathbf{x}^{2^p+1}, \dots, \mathbf{x}^{2^{(2^p)}-1}$ is the second block of size 2^p , and so on. At the end of each block we check inequalities (2.2) and (2.3). If they are satisfied and $2^p \geq \left\lceil 8\rho\sqrt{\frac{L}{t}} \right\rceil$, then by Lemma 2.1 we know that the residual of the function is divided in half; however if any of these inequalities fails, then as mentioned in the previous section we need to take “correction step” for the next block of iterates. The correction step is basically computing the next block of iterates in a way that satisfaction of inequalities (2.2) and (2.3) is guaranteed at the end of this block. Then the correction step is omitted in the subsequent blocks until the inequalities are violated again.

Recall that in an ordinary step, the new iterate, \mathbf{x}^{j+1} is calculated by a 2-dimensional search on the plane passing through \mathbf{x}^j , and parallel to the 2-dimensional subspace spanned by \mathbf{g}^j , and \mathbf{d}^j . Suppose at least one of the inequalities (2.2) and (2.3) is violated for k th block of p ; i.e. for the block of iterates $\mathbf{x}^{r_p}, \dots, \mathbf{x}^{r_p+2^p-1}$ where $r_p = (k-1)2^p$. Then for the next block we search for the new iterate \mathbf{x}^{j+1} on the space of $\mathbf{x}^j + S_p \{ \mathbf{g}^j, \mathbf{d}^j, \mathbf{q}_p^j, \mathbf{x}^j - \mathbf{x}^{r_p} \}$ where $\mathbf{q}_p^j = \sum_{i=r_p}^j \lambda^i \mathbf{g}^i$.

Finding the new iterate \mathbf{x}^{j+1} through a search on the space that in addition to \mathbf{g}^j and \mathbf{d}^j includes \mathbf{q}_p^j and $\mathbf{x}^j - \mathbf{x}^{r_p}$ is what we referred to as “correction step”. Notice that for each p with the violated constraints we increase the dimension of the search space by 2. However, the dimension of the search space never exceeds $O(P_u) = 2 + 2\lceil \log_2 j \rceil$, which happens to be the case when the inequalities are violated for all possible values of p .

It is quite easy to see that inequalities (2.2) and (2.3) are satisfied for the $(k+1)$ th block of p when we take the correction step throughout it. By KKT condition, we have $\langle \mathbf{g}^j, \mathbf{x}^j - \mathbf{x}^{r_p} \rangle = 0$ for all j in this block. Using this, along with the fact that $f(\mathbf{x}^j) < f(\mathbf{x}^{r_p})$ and non-negativity of λ^j for all j , we derive (2.2). Similarly one can argue that by KKT $\langle \mathbf{g}^j, \mathbf{q}_p^{j-1} \rangle = 0$ for all j , hence

$$\left\| \sum_{i=r_p}^{r_p+2^p-1} \lambda^i \mathbf{g}^i \right\| = \sqrt{\sum_{i=r_p}^{r_p+2^p-1} (\lambda^i)^2 \|\mathbf{g}^i\|^2},$$

which means inequality (2.3) is satisfied. After finding the iterates of one block through a correction step, the algorithm switches back to taking a regular step until the next failure of the inequalities. We can now present the algorithm in its entirety.

ALGORITHM 1.

```

 $\mathbf{x}^0 = \text{arbitrary}; \quad \rho : \text{given}; \quad S = \emptyset.$ 
for  $j = 1, 2, \dots$ 
   $\mathbf{x}^{j+1} = \operatorname{argmin}_{\mathbf{x} \in E^j} f(\mathbf{x}) \quad \text{where } E^j = \mathbf{x}^j + S_p \{ \mathbf{g}^j, \mathbf{d}^j, \cup_{p \in S} \mathbf{q}_p^j, \cup_{p \in S} \mathbf{x}^j - \mathbf{x}^{r_p} \}$ 
  Let  $\lambda^j = \sqrt{\frac{f(\mathbf{x}^j) - f(\mathbf{x}^{j+1})}{\|\mathbf{g}^j\|^2}}$ 
  for  $p = P_l, \dots, \lceil \log_2 j \rceil$ 
    if  $j+1 = k_p 2^p$  for some integer  $k_p$ 
      Let  $r_p = (k_p - 1) 2^p$ 
      if  $p \in S$ 
         $S = S \setminus \{p\}$ 
      else (i.e. if  $p \notin S$ ), check

```


$$\frac{(f(\mathbf{x}^{j+1}) - f(\mathbf{x}^{r_p}))}{4} \left(\sum_{i=r_p}^j \lambda^i \right) + \sum_{i=r_p}^j \lambda^i \langle \mathbf{g}^i, \mathbf{x}^i - \mathbf{x}^{r_p} \rangle < 0$$
 and

$$\left\| \sum_{i=r_p}^j \lambda^i \mathbf{g}^i \right\| \leq \rho \sqrt{\sum_{i=r_p}^j (\lambda^i)^2 \|\mathbf{g}^i\|^2}$$
 if any of the above inequalities fails, $S = S \cup \{p\}$
 end
 end
 end
 end

THEOREM 2.2. Suppose $m \geq \left\lceil 8\rho\sqrt{\frac{L}{t}} \right\rceil$, and \mathbf{x}^j is a sequence generated by algorithm 1 for solving problem (2.1); then $v_f(\mathbf{x}^{(n+4)m}) \leq \frac{1}{2}v_f(\mathbf{x}^{nm})$.

Proof. Let $\bar{p} \leq P$ be the integer for which $2^{\bar{p}-1} \leq m \leq 2^{\bar{p}}$; and let $s_{\bar{p}}$ stand for $2^{\bar{p}}$. Using algorithm 1, we are guaranteed that for at least one of any two consecutive blocks of size $s_{\bar{p}}$ inequalities (2.2) and (2.3) are satisfied. The size of this block is $s_{\bar{p}} \geq m \geq \left\lceil 8\rho\sqrt{\frac{L}{t}} \right\rceil$ and hence by lemma 2.1 we have

$$v_f(\mathbf{x}^{nm+2s_{\bar{p}}}) \leq \frac{1}{2}v_f(\mathbf{x}^{nm}). \quad (2.11)$$

Since $2s_{\bar{p}} \leq 4m$, so $f(\mathbf{x}^{nm+4m}) \leq f(\mathbf{x}^{nm+2s_{\bar{p}}})$; hence

$$v_f(\mathbf{x}^{nm+4m}) \leq v_f(\mathbf{x}^{nm+2s_{\bar{p}}}). \quad (2.12)$$

(2.11) and (2.12) gives us the result we wanted to show. \square

In our experiment with the algorithm, however, we only include the $\mathbf{x}^j - \mathbf{x}^{r_p}$ term in the correction step. We will discuss this and few other remarks on the implementation of the algorithm in the following section.

3. Implementation.

3.1. Solving Each Iteration. Obviously the most important part in CGSO is solving the subspace optimization problem in each iteration. In our implementation of the algorithm, we used Newton's method for this task unless it fails to rapidly converge to the optimum. In the case of failure of Newton's algorithm, the ellipsoid method carries out the task of solving the optimization problem. In other words we assign an upper bound to the number of iterations that Newton's method may take, and if it fails to converge within the given number of iterations the algorithm switches to the ellipsoid method for finding the next iterate.

Recall that at $(j+1)$ th iterate, we search for \mathbf{x}^{j+1} in the space of vectors $\mathbf{x} = \mathbf{x}^j + \alpha \mathbf{g}^j + \beta \mathbf{d}^j + \mathbf{a}Q + \mathbf{b}R$, where $Q \in \mathbb{R}^{n \times |S|}$ is the matrix formed by columns \mathbf{q}_p^j for all $p \in S$; R is the matrix of the same dimension with columns $\mathbf{x}^j - \mathbf{x}^{r_p}$ for all $p \in S$; $\alpha, \beta \in \mathbb{R}$, and $\mathbf{a}, \mathbf{b} \in \mathbb{R}^{|S|}$ are coefficients that we want to find. Let \mathbf{y} denote the variable of the subspace optimization problem, i.e., $\mathbf{y} = [\alpha, \beta, \mathbf{a}^t, \mathbf{b}^t]^t$; in addition let $B = [\mathbf{g}^j, \mathbf{d}^j, Q, R]$ and $K = 2 + 2|S|$. We can now state the formal presentation of the subspace optimization problem,

$$\min_{\mathbf{y} \in \mathbb{R}^K} f(\mathbf{x}^j + B\mathbf{y}) \quad (3.1)$$

As mentioned above, we solve problem (3.1) with Newton's method. Letting $\tilde{f}(\mathbf{y}) = f(\mathbf{x}^j + B\mathbf{y})$ and using chain rule we get the following formulas for the gradient

and Hessian of each Newton's iteration,

$$\nabla \tilde{f}(\mathbf{y}) = B^t \nabla f(\mathbf{x}) \quad (3.2)$$

$$\nabla^2 \tilde{f}(\mathbf{y}) = B^t \nabla^2 f(\mathbf{x}) B \quad (3.3)$$

Notice that some second order information of the function comes into play in equation (3.3); and we believe that this is one of the reasons that CGSO stands out in practice. We compute $\nabla f(\mathbf{x})$ and $\nabla^2 f(\mathbf{x})$ directly when $f(\mathbf{x})$ is simple enough. For more complicated functions we use automatic differentiation (AD) in backward mode to compute $\nabla f(\mathbf{x})$ and $\nabla^2 f(\mathbf{x})B$. Let $B^{(k)}$ denote k th column of matrix B . Backward AD enables us to keep the computational cost of $\nabla f(\mathbf{x})$ within a constant factor of the objective function evaluation cost; and the cost of computing $\nabla^2 f(\mathbf{x})B^{(k)}$ within a constant factor of the computational cost of gradient evaluation. The storage space required in backward AD, however, is more than the required storage in forward AD; and in worst case it can be proportional to the number of operations required for computing $f(\mathbf{x})$. In spite of that, we are able to keep the storage required by backward AD for the class of problems we studied in $O(s)$, where s is the space required to compute $f(\mathbf{x})$. Details on our test problems are presented in section 3.2. For more information on AD, one may refer to [12].

In addition to the storage required by AD, we need to store \mathbf{x}^j , and matrix B ; we also need to update and store \mathbf{x}^{r_p} , $\sum_{i=r_p}^j \lambda^i$, $\sum_{i=r_p}^j \lambda^i \langle \mathbf{g}^i, \mathbf{x}^i - \mathbf{x}^{r_p} \rangle$, $\sum_{i=r_p}^j \lambda^i \mathbf{g}^i$, $\sum_{i=r_p}^j (\lambda^i)^2 \|\mathbf{g}^i\|^2$ for all $p \in \{P_1, \dots, \log_2 j\}$. In our experiment with CGSO we take P_1 to be 4. We find that K is equal to 2 in every case except one, in which it reaches the value of 3. Hence the required storage space for the above elements is in $O(n \max\{K, \log_2 j\})$.

A difficulty we need to overcome in solving a problem with CGSO, especially when we are trying to achieve high accuracy, is round-off error. In particular $f(\mathbf{x}^j) - f(\mathbf{x}^{j+1})$, which is required for computing λ^j , gets more and more inaccurate as the iterates approach the optimum. To overcome this, we took advantage of the second order Taylor series expansion. We have a subroutine that analyze the absolute error of $f(\mathbf{x}^j) - f(\mathbf{x}^{j+1})$ computed directly and through Taylor series, i.e. $f(\mathbf{x}^j) - f(\mathbf{x}^{j+1}) \approx -(\nabla f(\mathbf{x}^j)^t (\mathbf{x}^{j+1} - \mathbf{x}^j) + \frac{1}{2} (\mathbf{x}^{j+1} - \mathbf{x}^j)^t \nabla^2 f(\mathbf{x}^j) (\mathbf{x}^{j+1} - \mathbf{x}^j)$; the one with smaller error is accepted. In some cases the error analysis is not easy, hence a heuristic is used to choose the preferred formula. Computing the difference of two objective values is appeared in inequality (2.2) as well, in $f(\mathbf{x}^{m-1}) - f(\mathbf{x}^0)$; the same subroutine is used to compute this term.

3.2. Numerical Results. We have tested our algorithm on the following classes of problem:

- $f_1(\mathbf{x}) = -\sum_{i=1}^m \log(\mathbf{a}_i^t \mathbf{x} - \mathbf{b}_i)$
- $f_2(\mathbf{x}) = -\mathbf{c}^t \mathbf{x} - \log(\det(C - \text{Diag}(\mathbf{x})))$
- $f_3(\mathbf{x}) = \sum_{i=1}^m (\mathbf{a}_i^t \mathbf{x} - \mathbf{b}_i)^d$ where d is a given even integer.

Functions f_1 and f_2 are log-barrier functions, and f_3 is an approximation to the infinity norm of the vector $A\mathbf{x} - \mathbf{b}$. Note that we need to restrict the degree, d , to even numbers to save the convexity of f_3 .

All the codes for CGSO, Ellipsoid method, and automatic differentiation are written in MATLAB. The Hessian of f_1 and f_3 can be derived easily in closed form. The Hessian of f_2 was obtained by applying backward mode AD by hand to a program that computes f_2 .

The stopping criteria we used for both CGSO and its subproblems are $\|\nabla f(\mathbf{x})\| \leq \epsilon$ and $\|\nabla \tilde{f}(\mathbf{x})\| \leq \epsilon_N$, respectively. In our implementation, ϵ_N at iteration j is $\frac{\|B^t \nabla f(\mathbf{x}^j)\|}{100}$; however, if the subproblem is solved by Newton's method, the obtained solution is usually more accurate due to the fast local convergence of the Newton's method. Notice that for f_1 and f_2 we have some hidden constraints, namely $\mathbf{a}_i^t \mathbf{x} - \mathbf{b}_i > 0$ in f_1 and $C - \text{Diag}(\mathbf{x}) \succ 0$ in f_2 . Therefore CGSO switches to ellipsoid method if Newton's method does not converge in 15 iterations or if the iterates get infeasible.

In [7], Hager and Zhang compared their variant of CG with L-BFGS and PRP₊, and established the superior performance of their variant of CG. Hence, we compare our algorithm with their variant of CG, represented by CG_{HZ} in Table 3.1 that summarizes this comparison. The top portion of the table explains the instances that we generated randomly. Parameters m , n , d , and ϵ are defined as above; m_f for f_2 represents $\mathbf{c} = m_f \mathbf{e}$ where \mathbf{e} is the all ones vector. The point of this parameter is that larger values of m_f force $C - \text{Diag}(\mathbf{x}^*)$ for the optimizer x^* closer to the boundary of the feasible region, hence the problem become more ill-conditioned.

Let A be the m by n matrix formed by \mathbf{a}_i^t as its rows; “ds.” indicates the density, the percentage of nonzero entries of A for f_1 and f_3 , and density of C for f_2 . “It. count” for both CG_{HZ} and CGSO indicates the total number of major iteration the algorithm takes before convergence. “LS. count” is the total iteration performed by the line search routine in CG_{HZ}; for each iterate of LS we need one gradient evaluation. In CGSO, one gradient and one Hessian evaluation is required for each Newton's iterate; and one gradient is calculated in each iteration of the ellipsoid algorithm. As table 3.1 shows, the number of iterations CGSO takes is less than CG_{HZ} in all instances, and the number of iterations for solving subproblems is significantly less than the number of line search iterations; especially for instances 3 and 4, this difference is considerable. The last row of the table implies that no correction step was taken in any of these instances, except one. Actually the purpose of “Ins. 7” is to show that correction step might be required for some problems. In this instance A is a square matrix of size 50×50 with $O(10^3)$ condition number. The correction step is taken throughout 7 blocks corresponding to P_l , for which the size of the subproblems reaches 3. This, however, is consistent with the theory because the larger the condition number is, the larger $\frac{L}{t}$ is; therefore 2^{P_l} may not be a good approximation of $\lceil 8\rho\sqrt{\frac{L}{t}} \rceil$.

3.3. Parameter ρ . Recall that ρ is a parameter of CGSO required for checking inequality (2.3). As mentioned before, we do not check inequality (2.3) in our implementation of the algorithm; instead we gather the values of ρ and study their trend. Table 3.2 summarizes the maximum value ρ reached in each instance over all values of p . Figure 3.1 and 3.2 depict ρ for Ins. 1 and Ins. 3, respectively. We chose these instances because they have higher ρ_{max} and iteration count in each category. Our observation suggests that a reasonably small bound for ρ should suffice. Notice that the plotted values imply that ρ reaches its maximum for some p and then it decreases for higher values of p , so it does not grow with p . Furthermore, as we get closer to the optimum parameter ρ gets closer to 1. This is a common pattern for all test problems we had. Instead of fixing ρ one may adaptively update this parameter. In other words, we can assign a value to ρ and if inequality (2.3) fails for more than a certain number of blocks, then we increase ρ , and as the algorithm progresses towards the optimum we can decrease ρ .

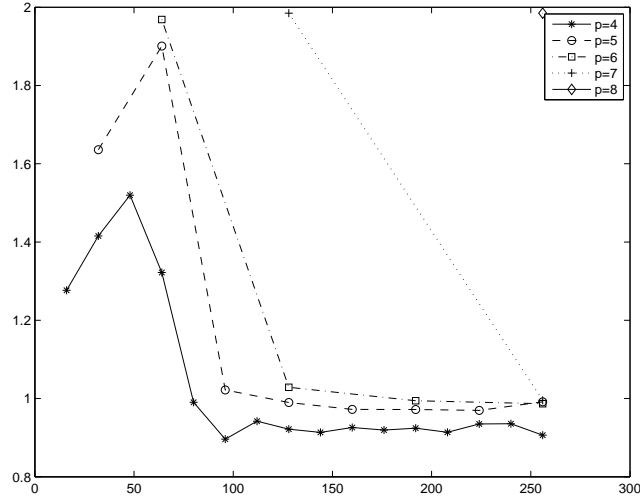
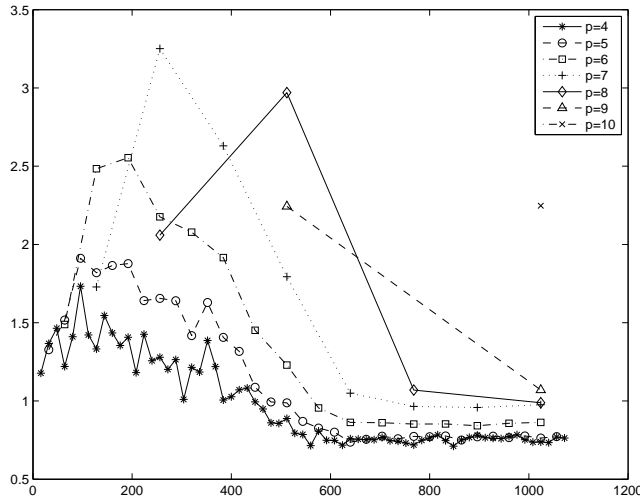
TABLE 3.1
Comparison of CGSO and CG_{HZ}

		f_1		f_2		f_3		
		Ins. 1	Ins. 2	Ins. 3	Ins. 4	Ins. 5	Ins. 6	Ins. 7
m		6000	6000	-	-	1500	2500	50
n		2000	2000	500	1000	3000	5000	50
m_f		-	-	100	10	-	-	-
d		-	-	-	-	6	4	4
$ds.$		1	0.5	0.012	0.006	0.5	0.5	1
ϵ		1e-8	1e-8	1e-3	1e-8	1e-18	1e-18	1e-8
CG _{HZ}	It. count	404	357	3059	928	225	203	3442
	LS. count	2999	2535	92682	14596	6220	7333	53474
CGSO	It. count	261	213	1080	419	148	136	658
	Newton	402	323	1884	600	545	0	1055
	ellipsoid	394	422	6195	1641	410	0	0
	Corr.	0	0	0	0	0	0	7

TABLE 3.2
Parameter ρ

	Ins. 1	Ins. 2	Ins. 3	Ins. 4	Ins. 5	Ins. 6	Ins. 7
ρ_{max}	1.9848	1.9803	3.2511	2.144	1.016	1.0281	1.9430

4. Conclusion. We presented CGSO for solving unconstrained strongly convex functions. CGSO is a variant of CG, since the update step in each iteration is a combination of previous gradients, and it reduces to linear CG when applied to a quadratic function. The coefficients of previous gradients, however, do not follow an updating rule and are found by a subspace optimization subproblem. We have discussed that these subproblems are mostly two dimensional; in some cases the dimension of the subproblems may reach slightly higher values, but it certainly never exceeds $O(\log j)$, where j is the iteration count. We have also shown that CGSO benefits from the optimal complexity bound of $O\left(\sqrt{\frac{L}{\epsilon}} \log\left(\frac{1}{\epsilon}\right)\right)$ in theory. CGSO does not depend on any prior information about the function and can easily be implemented. In practice, it outperforms the variant of CG proposed by Hager and Zhang [7] which is known to be stronger than other techniques such as L-BFGS or PRP₊. Practical efficiency of CGSO can be improved by incorporating some second order information of the function in solving the subproblems with Newton's method as discussed in the previous section.

FIG. 3.1. Parameter ρ for Ins. 1FIG. 3.2. Parameter ρ for Ins. 3

REFERENCES

- [1] M. Al-Baali. Descent property and global convergence of the fletcher-reeves method with inexact line search. *IMA Journal of Numerical Analysis*, 5:121–124, 1985.
- [2] Y. H. Dai and Y. Yuan. A nonlinear conjugate gradient method with a strong global convergence property. *Siam Journal on Optimization*, 10(1):177–182, 1999.
- [3] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *Computer Journal*, 7:149–154, 1964.
- [4] J. C. Gilbert and J. Nocedal. Global convergence properties of conjugate gradient methods for optimization. *Siam Journal on Optimization*, 2(1):21–42, 1992.
- [5] G. H. Golub and C. F. Van loan. *Matrix Computations*. Hopkins Fulfillment Service, 1996.
- [6] W. W. Hager and H. Zhang. A survey of nonlinear conjugate gradient methods.
- [7] W. W. Hager and H. Zhang. A new conjugate gradient method with guaranteed descent and an efficient line search. *SIAM Journal on Optimization*, 16(1):170–192, 2005.
- [8] M. R. Hestenes and E. Stiefel. Methods of conjugate gradient for solving linear systems. *Journal*

- of Research of the National Bureau of Standards*, 49:409–436, 1952.
- [9] A. S. Nemirovsky and D. B. Yudin. *Problem Complexity and Method Efficiency in Optimization*. John Wiley & sons, 1983.
 - [10] Y. E. Nesterov. A method of solving a convex programming problem with convergence rate $o(\frac{1}{k^2})$. *Soviet mathematics, Doklady*, 27(2):372–376, 1983.
 - [11] Y. E. Nesterov. Smooth minimization of nonsmooth functions. *Math. Programming*, 103:127–152, 2005.
 - [12] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Science, 2006.
 - [13] E. Polak and G. Ribière. Note sur la convergence de méthodes de directions conjuguées. *Revue Francaise d’informatique et de Recherche Opérationnelle*, 16:35–43, 1969.
 - [14] Boris T. Polyak. *Introduction to Optimization*. Optimization Software Inc.,, 1987.
 - [15] P. Tseng. On accelerated proximal gradient methods for convex-concave optimization. 2008.
 - [16] Y. Yuan and J. Stoer. A subspace study on conjugate gradient algorithms. *ZAMM (Zeitschrift fr angewandte Mathematik und Mechanik)*, 11:69–77, 1995.